SHACL – Shapes Constraint Language in AllegroGraph

SHACL is a SHApe Constraint Language. It specifies a vocabulary (using triples) to describe the shape that data should have. The shape specifies things like the following simple requirements:

- How many triples with a specified subject and predicate should be in the repository (e.g. at least 1, at most 1, exactly 1).
- What the nature of the object of a triple with a specified subject and predicate should be (e.g. a string, an integer, etc.)

See the specification for more examples.

SHACL allows you to validate that your data is conforming to desired requirements.

For a given validation, the shapes are in the Shapes Graph (where graph means a collection of triples) and the data to be validated is in the Data Graph (again, a collection of triples). The SHACL vocabulary describes how a given shape is linked to targets in the data and also provides a way for a Data Graph to specify the Shapes Graph that should be used for validation. The result of a SHACL validation describes whether the Data Graph conforms to the Shapes Graph and, if it does not, describes each of the failures.

Namespaces Used in this Document

Along with standard predefined namespaces (such as rdf: for <http://www.w3.org/1999/02/22-rdf-syntax-ns#> and rdfs: for <http://www.w3.org/2000/01/rdf-schema#>), the following are used in code and examples below:
A Simple Example

Suppose we have a *Employee* class and for each *Employee* instance, there must be exactly one triple of the form

emp001 hasID "000-12-3456"

where the object is the employee’s ID Number, which has the format is [3 digits]-[2 digits]-[4 digits].

This TriG file encapsulates the constraints above:

```trig
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<https://franz.com#Shapes> {  
  <https://franz.com#EmployeeShape>  
    a sh:NodeShape ;  
    sh:targetClass <https://franz.com#Employee> ;  
    sh:property [  
      sh:path <https://franz.com#hasID> ;  
      sh:minCount 1 ;  
      sh:maxCount 1 ;  
      sh:datatype xsd:string ;  
      sh:pattern "^[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9][0-9]$" ;  
    ] .
}
```

It says that for instances of fr:Employee (sh:targetClass <https://franz.com#Employee>), there must be exactly 1 triple with predicate (path) fr:hasID and the object of that triple must be a string with pattern [3 digits]-[2 digits]-[4 digits] (sh:pattern "^[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9][0-9][0-9]$").
This TriG file defines the Employee class and some employee instances:

```trig
@prefix fr: <https://franz.com#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

{  
  fr:Employee  
    a rdfs:Class .  
  fr:emp001  
    a fr:Employee ;  
    fr:hasID "000-12-3456" ;  
    fr:hasID "000-77-3456" .  
  fr:emp002  
    a fr:Employee ;  
    fr:hasID "00-56-3456" .  
  fr:emp003  
    a fr:Employee .  
}
```

Recalling the requirements above, we immediately see these problems with these triples:

1. `emp001` has two `hasID` triples.
2. The value of `emp002`'s ID has the wrong format (two leading digits rather than 3).
3. `emp003` does not have a `hasID` triple.

We load the two TriG files into our repository, and end up with the following triple set. Note that all the employee triples use the default graph and the SHACL-related triples use the graph <https://franz.com#Shapes> specified in the TriG file.
Now we use **agtool shacl-validate** to validate our data:

```
bin/agtool shacl-validate --data-graph default --shapes-graph https://franz.com#Shapes shacl-repo-1
```

Validation report: Does not conform

Created: 2019-06-27T10:24:10

Number of shapes graphs: 1
Number of data graphs: 1
Number of NodeShapes: 1
Number of focus nodes checked: 3

3 validation results:

Result:

Focus node: <https://franz.com#emp001>
Path: <https://franz.com#hasID>
Source Shape: _:b7A1D241Ax1

Constraint Component: <http://www.w3.org/ns/shacl#MaxCountConstraintComponent>
Severity: <http://www.w3.org/ns/shacl#Violation>

Result:

Focus node: <https://franz.com#emp002>
Path: <https://franz.com#hasID>
Value: "00-56-3456"
Source Shape: _:b7A1D241Ax1

Constraint Component: <http://www.w3.org/ns/shacl#PatternConstraintComponent>
Severity: <http://www.w3.org/ns/shacl#Violation>

Result:
The validation fails with the problems listed above. The Focus node is the subject of a triple that did not conform. Path is the predicate or a property path (predicates in this example). Value is the offending value. Source Shape is the shape that established the constraint (you must look at the shape triples to see exactly what Source Shape is requiring).

We revise our employee data with the following SPARQL expression, deleting one of the emp001 triples, deleting the emp002 triple and adding a new one with the correct format, and adding an emp003 triple.

```
prefix fr: <https://franz.com#>

DELETE DATA {fr:emp002 fr:hasID "00-56-3456" } ;

INSERT DATA {fr:emp002 fr:hasID "000-14-1772" } ;

DELETE DATA {fr:emp001 fr:hasID "000-77-3456" } ;

INSERT DATA {fr:emp003 fr:hasID "000-54-9662" } ;
```

Now our employee triples are

<table>
<thead>
<tr>
<th>s</th>
<th>p</th>
<th>o</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp002</td>
<td>hasID</td>
<td>&quot;000-14-1772&quot;</td>
<td></td>
</tr>
<tr>
<td>emp003</td>
<td>hasID</td>
<td>&quot;000-54-9662&quot;</td>
<td></td>
</tr>
<tr>
<td>Employee</td>
<td>rdf:type</td>
<td>rdfs:Class</td>
<td></td>
</tr>
<tr>
<td>emp001</td>
<td>rdf:type</td>
<td>Employee</td>
<td></td>
</tr>
<tr>
<td>emp001</td>
<td>hasID</td>
<td>&quot;000-12-3456&quot;</td>
<td></td>
</tr>
<tr>
<td>emp002</td>
<td>rdf:type</td>
<td>Employee</td>
<td></td>
</tr>
<tr>
<td>emp003</td>
<td>rdf:type</td>
<td>Employee</td>
<td></td>
</tr>
</tbody>
</table>

We run the validation again and are told our data conforms:

```
% bin/agtool shacl-validate --data-graph default --shapes-graph https://franz.com#Shapes shacl-repo-1
```
### SHACL API

The example above illustrates the SHACL steps:

1. Have a data set with triples that should conform to a shape
2. Have SHACL triples that express the desired shape
3. Run SHACL validation to determine if the data conforms

Note that SHACL validation does not modify the data being validated. Once you have the conformance report, you must modify the data to fix the conformance problems and then rerun the validation test.

The main entry point to the API is `agtool shacl-validate`. It takes various options and has several output choices. Online help for `agtool shacl-validate` is displayed by running `agtool shacl-validate --help`.

In order to validate triples, the system must know:

1. What triples to examine
2. What rules (SHACL triples) to use
3. What to do with the results

### Specifying what triples to examine

Two arguments to `agtool shacl-validate` specify the triples to evaluate: `--data-graph` and `--focus-node`. Each can be specified
multiple times.

- The --data-graph argument specifies the graph value for triples to be examined. Its value must be an IRI or default. Only triples in the specified graphs will be examined. default specifies the default graph. It is also the default value of the --data-graph argument. If no value is specified for --data-graph, only triples in the default graph will be examined. If a value for --data-graph is specified, triples in the default graph will only be examined if --data-graph default is also specified.

- The --focus-node argument specifies IRIs which are subjects of triples. If this argument is specified, only triples with these subjects will be examined. To be examined, triples must also have graph values specified by --data-graph arguments. --focus-node does not have a default value. If unspecified, all triples in the specified data graphs will be examined. This argument can be specified multiple times.

The --data-graph argument was used in the simple example above. Here is how the --focus-node argument can be used to restrict validation to triples with subjects <https://franz.com#emp002> and <https://franz.com#emp003> and to ignore triples with subject <https://franz.com#emp001> (applying agtool shacl-validate to the orignal non-conformant data):

% bin/agtool shacl-validate --data-graph default \  --shapes-graph https://franz.com#Shapes \  --focus-node https://franz.com#emp003 \  --focus-node https://franz.com#emp002 shacl-repo-1
Validation report: Does not conform
Created: 2019-06-27T11:37:49
Number of shapes graphs: 1
Number of data graphs: 1
Number of NodeShapes: 1
Number of focus nodes checked: 2
2 validation results:
Result:
Focus node: <https://franz.com#emp003>
Path: <https://franz.com#hasID>
Source Shape: _:b7A1D241Ax2
Constraint Component: <http://www.w3.org/ns/shacl#MinCountConstraintComponent>
Severity: <http://www.w3.org/ns/shacl#Violation>

Result:
Focus node: <https://franz.com#emp002>
Path: <https://franz.com#hasID>
Value: "00-56-3456"
Source Shape: _:b7A1D241Ax2
Constraint Component: <http://www.w3.org/ns/shacl#PatternConstraintComponent>
Severity: <http://www.w3.org/ns/shacl#Violation>

Specifying What Shape Triples to Use

Two arguments to agtool shacl-validate, analogous to the two arguments for data described above, specify Shape triples to use. Further, following the SHACL spec, data triples with predicate <http://www.w3.org/ns/shacl#shapeGraph> also specify graphs containing Shape triples to be used.

The arguments to agtool shacl-validate are the following. Each may be specified multiple times.

- The --shapes-graph argument specifies the graph value for shape triples to be used for SHACL validation. Its value must be an IRI or default. default specifies the default graph. The --shapes-graph argument has no default value. If unspecified, graphs specified by data triples with the <http://www.w3.org/ns/shacl#shapeGraph> predicate will be used (they are used whether or not --shapes-graph has a value). If --shapes-graph has no value and there are no data triples with
the <http://www.w3.org/ns/shacl#shapeGraph> predicate, the data graphs are used for shape graphs. (Shape triples have a known format and so can be identified among the data triples.)

- The --shape argument specifies IRIs which are subjects of shape nodes. If this argument is specified, only shape triples with these subjects and subsidiary triples to these will be used for validation. To be included, the triples must also have graph values specified by the --shapes-graph arguments or specified by a data triple with the <http://www.w3.org/ns/shacl#shapeGraph> predicate. --shape does not have a default value. If unspecified, all shapes in the shapes graphs will be used.

**Other APIs**

There is a lisp API using the function validate-data-graph, defined next:

```lisp
validate-data-graphdb &key data-graph-iri/s shapes-graph-iri/s shape/s focus-node/s verbose conformance-only?
```

Perform SHACL validation and return a validation-report structure.

The validation uses data-graph-iri/s to construct the dataGraph. This can be a single IRI, a list of IRIs or NIL, in which case the default graph will be used. The shapesGraph can be specified using the shapes-graph-iri/s parameter which can also be a single IRI or a list of IRIs. If shape-graph-iri/s is not specified, the SHACL processor will first look to create the shapesGraph by finding triples with the predicate sh:shapeGraph in the dataGraph. If there are no such triples, then the shapesGraph will be assumed to be the same as the dataGraph.
Validation can be restricted to particular shapes and focus nodes using the shape/s and focus-node/s parameters. Each of these can be an IRI or list of IRIs.

If conformance-only? is true, then validation will stop as soon as any validation failures are detected.

You can use validation-report-conforms-p to see whether or not the dataGraph conforms to the shapesGraph (possibly restricted to just particular shape/s and focus-node/s).

The function validation-report-conforms-p returns t or nil as the validation struct returned by validate-data-graph does or does not conform.

validation-report-conforms-preport function

Returns t or nil to indicate whether or not REPORT (a validation-report struct) indicates that validation conformed.

There is also a REST API. See HTTP reference.

**Validation Output**

The simple example above and the SHACL examples below show output from `agtool validate-shacl`. There are various output formats, specified by the --output option. Those examples use the plain format, which means printing results descriptively. Other choices include json, trig, trix, turtle, nquads, rdf-n3, rdf/xml, and ntriples. Here are the simple example (uncorrected) results using ntriples output:

```
% bin/agtool shacl-validate --output ntriples --data-graph default --shapes-graph https://franz.com#Shapes shacl-repo-1
_:b271983AAx1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/ns/shacl#ValidationReport> .
```
You can have the triples added to the repository by specifying the --add-to-repo option true.

In the plain output information is provided about how many data graphs are examined, how many shape graphs were specified and node shapes are found, and how many focus nodes are checked. If zero focus nodes are checked, that is likely not what you want and something has gone wrong. Here we mis-spell the name of the shape graph (https://franz.com#shapes instead of https://franz.com#Shapes) and get 0 focus nodes checked:

% bin/agtool shacl-validate --data-graph default --shapes-graph https://franz.com#shapes shacl-repo-1
Validation report: Conforms
Created: 2019-06-28T10:34:22
Number of shapes graphs: 1
Number of data graphs: 1
Number of NodeShapes: 0
Number of focus nodes checked: 0

**SPARQL integration**

There are two sets of magic properties defined: one checks for basic conformance and the other produces validation reports as triples:
In all of the above `?dataGraph` and `?shapesGraph` can be IRIs, the literal ‘default’, or a variable that is bound to a SPARQL collection (list or set) that was previously created with a function like https://franz.com/ns/allegrograph/6.5.0/fn#makeSPARQLList or https://franz.com/ns/allegrograph/6.5.0/fn#lookupRdfList. If a collection is used, then the SHACL processor will create a temporary RDF merge of all of the graphs in it to produce the data graph or the shapes graph.

Similarly, `?shapeOrShapeCollection` and `?nodeOrNodeCollection` can be bound to an IRI or a SPARQL collection. If a collection is used, then it must be bound to a list of IRIs. The SHACL
processor will restrict validation to the shape(s) and focus node(s) (i.e. nodes that should be validated) specified.

The shapesGraph argument is optional in both of the shaclConforms and shaclValidationReport magic properties. If the shapesGraph is not specified, then the shapesGraph will be created by following triples in the dataGraph that use the sh:shapesGraph predicate. If there are no such triples, then the shapesGraph will be the same as the dataGraph.

For example, the following SPARQL expression

construct { ?s ?p ?o } where {
    # form a collection of focusNodes
    bind(<https://franz.com/ns/allegrograph/6.6.0/fn#makeSPARQLList>(
        <http://Journal1/1942/Article25>,
        <http://Journal1/1943>) as ?nodes)
    (\s\sp\so)
    <https://franz.com/ns/allegrograph/6.6.0/shaclShapeValidationReport1>
        ('default' <ex://franz.com/documentShape1> ?nodes) .
}

would use the default graph as the Data Graph and the Shapes Graph and then validate two focus nodes against the shape <ex://franz.com/documentShape1>.

**SHAACL Example**

We build on our simple example above. Start with a fresh repository so triples from the simple example do not interfere with this example.

We start with a TriG file with various shapes defined on some classes.

@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix fr: <https://franz.com#> .
sh:hasValue "President" ;
]
[

## Must have a supervisor
sh:path fr:hasSupervisor ;
sh:minCount 1 ;
sh:maxCount 1 ;
sh:class fr:Employee ;
]
)

sh:or (  # Every employee must either have a wage or a salary
[  
  sh:path fr:hasSalary ;
  sh:datatype xsd:integer ;
  sh:minInclusive 3000 ;
  sh:minCount 1 ;
  sh:maxCount 1 ;
]
[
  sh:path fr:hasWage ;
  sh:datatype xsd:decimal ;
  sh:minExclusive 15.00 ;
  sh:minCount 1 ;
  sh:maxCount 1 ;
]
)
.

This file says the following about instances of the class fr:Employee:

1. Every employee must have exactly one ID (object of fr:hasID), a string of the form NNN-NN-NNNN where the Ns are digits (this is the simple example requirement).
2. Every employee must have exactly one fr:employeeType triple with value either “Manager” or “Worker”.
3. Employees may have a `fr:birthYear` triple, and if so, the value must be 2001 or earlier.
4. Employees must have a `fr:hasTitle` and may have more than one.
5. All employees except the one with title “President” must have a supervisor (specified with `fr:hasSupervisor`).
6. Every employee must either have a wage (a decimal specifying hourly pay, greater than 15.00) or a salary (an integer specifying monthly pay, greater than or equal to 3000).

Here is some employee data:

```prefix fr: <https://franz.com#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
{
    fr:Employee
    a rdfs:Class .

    fr:emp001
    a fr:Employee ;
    fr:hasID "000-12-3456" ;
    fr:hasTitle "President" ;
    fr:employeeType "Manager" ;
    fr:birthYear "1953"^^xsd:integer ;
    fr:hasSalary "10000"^^xsd:integer .

    fr:emp002
    a fr:Employee ;
    fr:hasID "000-56-3456" ;
    fr:hasTitle "Foreman" ;
    fr:employeeType "Worker" ;
    fr:birthYear "1966"^^xsd:integer ;
    fr:hasSupervisor fr:emp003 ;
    fr:hasWage "20.20"^^xsd:decimal .

    fr:emp003
    a fr:Employee ;
```
fr:hasID "000-77-3232";
fr:hasTitle "Production Manager";
fr:employeeType "Manager";
fr:birthYear "1968"^^xsd:integer;
fr:hasSupervisor fr:emp001;
fr:hasSalary "4000"^^xsd:integer.

fr:emp004
  a fr:Employee;
  fr:hasID "000-88-3456";
  fr:hasTitle "Fitter";
  fr:employeeType "Worker";
  fr:birthYear "1979"^^xsd:integer;
  fr:hasSupervisor fr:emp002;
  fr:hasWage "17.20"^^xsd:decimal.

fr:emp005
  a fr:Employee;
  fr:hasID "000-99-3492";
  fr:hasTitle "Fitter";
  fr:employeeType "Worker";
  fr:birthYear "2000"^^xsd:integer;
  fr:hasSupervisor fr:emp002;
  fr:hasWage "17.20"^^xsd:decimal.

fr:emp006
  a fr:Employee;
  fr:hasID "000-78-5592";
  fr:hasTitle "Filer";
  fr:employeeType "Intern";
  fr:birthYear "2003"^^xsd:integer;
  fr:hasSupervisor fr:emp002;
  fr:hasWage "14.20"^^xsd:decimal.

fr:emp007
  a fr:Employee;
  fr:hasID "000-77-3232";
  fr:hasTitle "Sales Manager";
  fr:hasTitle "Vice President";
  fr:employeeType "Manager";
  fr:birthYear "1962"^^xsd:integer;
  fr:hasSupervisor fr:emp001;
fr:hasSalary "7000"^^xsd:integer .
}

Comparing these data with the requirements, we see these problems:

1. emp005 does not have a supervisor.
2. emp006 is pretty messed up, with (1) employeeType “Intern”, not an allowed value, (2) a birthYear (2003) later than the required maximum of 2001, and (3) a wage (14.40) less than the minimum (15.00).

Otherwise the data seems OK.

We load these two TriG files into an empty repository (which we have named shacl-repo-2). We specify the default graph for the data and the https://franz.com#ShapesGraph for the shapes. (Though not required, it is a good idea to specify a graph for shape data as it makes it easy to delete and reload shapes while developing.) We have 101 triples, 49 data and 52 shape. Then we run agtool shacl-validate:

% bin/agtool shacl-validate --shapes-graph https://franz.com#ShapesGraph --data-graph default shacl-repo-2

There are four violations, as expected, one for emp005 and three for emp006.

Validation report: Does not conform
Created: 2019-07-03T11:35:27
Number of shapes graphs: 1
Number of data graphs: 1
Number of NodeShapes: 1
Number of focus nodes checked: 7

4 validation results:
Result:
Focus node: <https://franz.com#emp005>
Value: <https://franz.com#emp005>
Source Shape: <https://franz.com#EmployeeShape>
Fixing the data is left as an exercise for the reader.